covid_analysis

December 29, 2022

[1]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

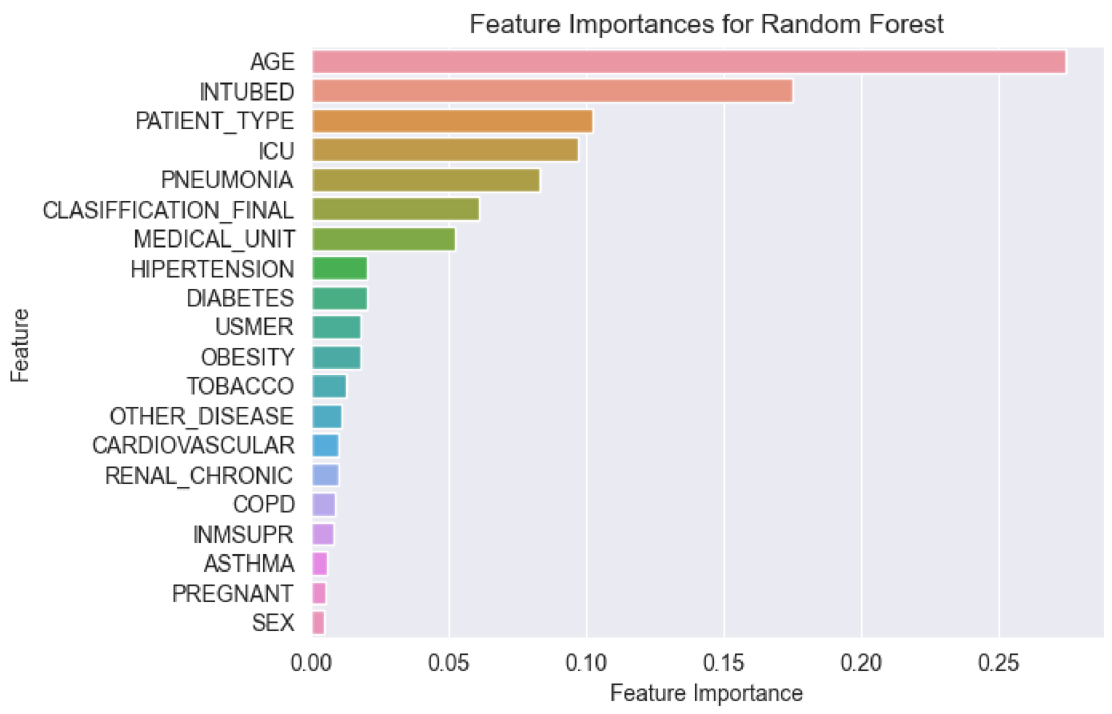
[2]:
```
df = pd.read_csv("Covid Data.csv")
df["DEATH"] = (df["DATE_DIED"] != "9999-99-99").astype(int)
X = df.drop(columns=["DATE_DIED", "DEATH"])
y = df["DEATH"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
accuracy_list, precision_list, recall_list, f1_list = [], [], [], []
```

[3]:
```
# Create random forest classifier
forest = RandomForestClassifier(n_estimators=110, random_state=42, max_depth=60)
forest.fit(X_train, y_train)
```

[3]:
```
RandomForestClassifier(max_depth=60, n_estimators=110, random_state=42)
```

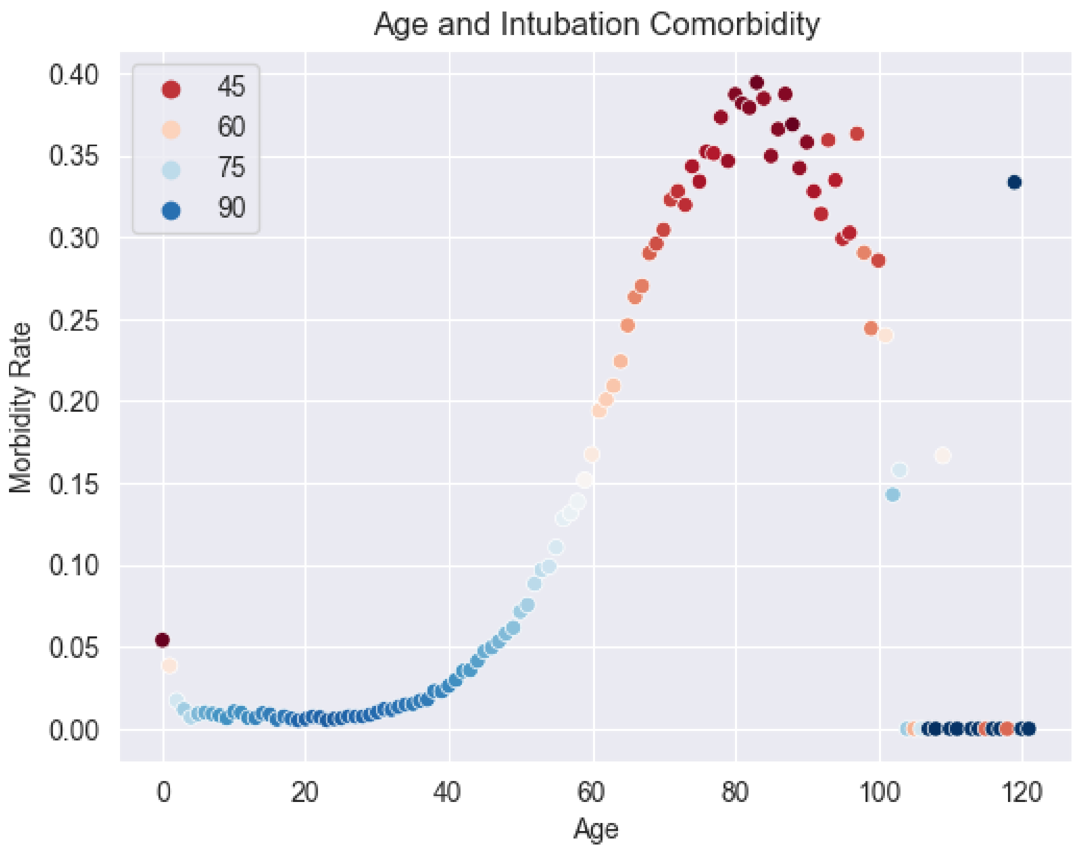
We begin with a random forest feature analysis.

[4]:
```
importances = forest.feature_importances_
feature_names = X.columns
indices = np.argsort(importances)[::-1]
sorted_feature_names = [feature_names[i] for i in indices]
sns.barplot(x=importances[indices], y=sorted_feature_names)
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.title("Feature Importances for Random Forest")
plt.savefig("feature_importances.png")
plt.show()
```

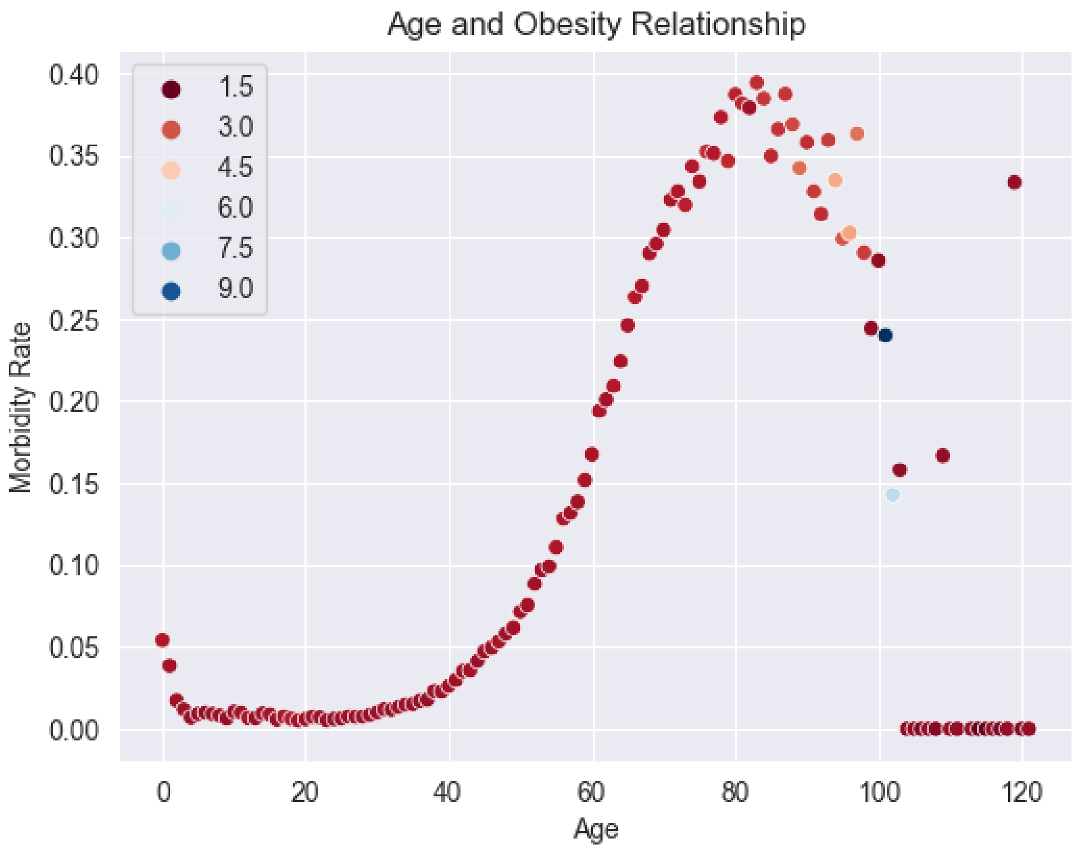


The two graphs below highlight age as a key factor for morbidity.

[5]:
```
grouped = df.groupby("AGE")
death_percentages = grouped["DEATH"].mean()
intubation_percentages = grouped["INTUBED"].mean()
sns.scatterplot(x=death_percentages.index, y=death_percentages.values,␣
  ↪hue=intubation_percentages.values, palette="RdBu")
plt.xlabel("Age")
plt.ylabel("Morbidity Rate")
plt.title("Age and Intubation Comorbidity")
plt.savefig("age_intubation_comorbidity.png")
plt.show()
```

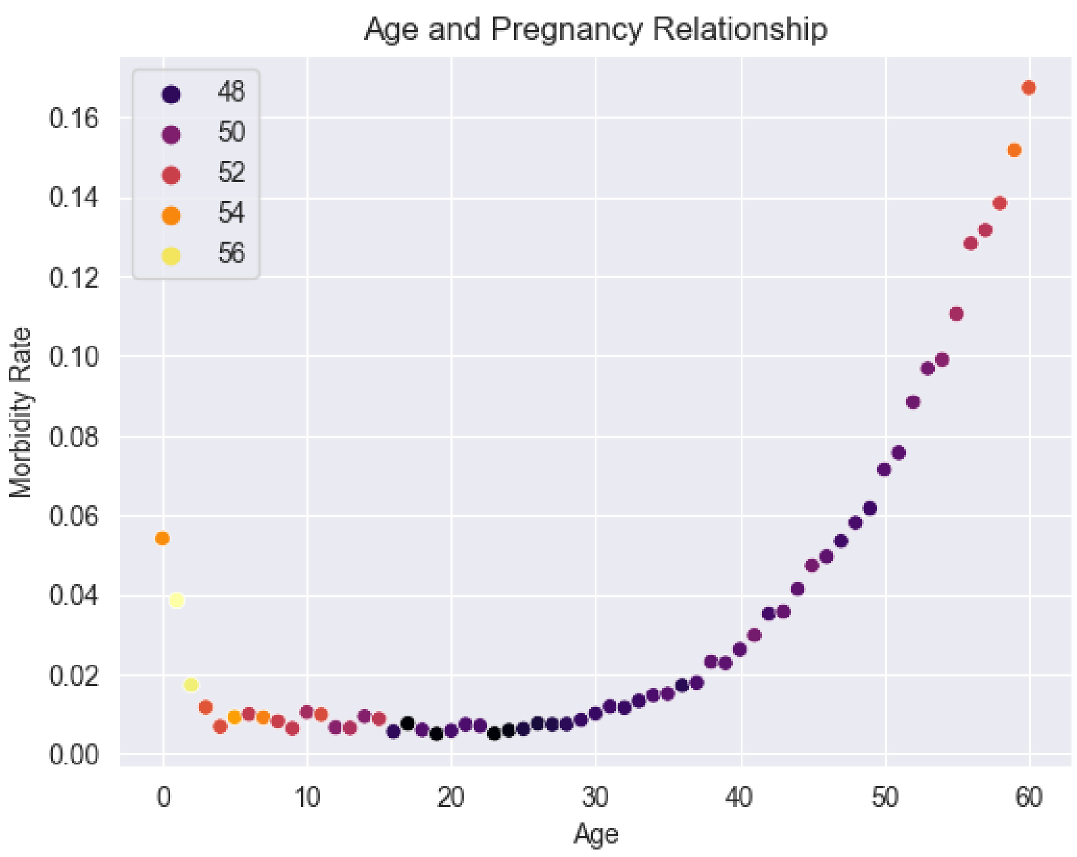


[6]:
```
grouped = df.groupby("AGE")
death_percentages = grouped["DEATH"].mean()
obesity_percentages = grouped["OBESITY"].mean()
sns.scatterplot(x=death_percentages.index, y=death_percentages.values,␣
  ↪hue=obesity_percentages.values, palette="RdBu")
plt.xlabel("Age")
plt.ylabel("Morbidity Rate")
plt.title("Age and Obesity Relationship")
plt.savefig("age_obesity_relationship.png")
plt.show()
```



This data set had dirty data when looking at pregnancies. There are 4 (or five) categories, but it is not made clear. Here is a cleaned representation.

[7]:
```
df = df.query("AGE <= 60")
grouped = df.groupby("AGE")
death_percentages = grouped["DEATH"].mean()
pregnancy_percentages = grouped["PREGNANT"].mean()
sns.scatterplot(x=death_percentages.index, y=death_percentages.values,␣
 ↪hue=pregnancy_percentages.values, palette="inferno")
plt.xlabel("Age")
plt.ylabel("Morbidity Rate")
plt.title("Age and Pregnancy Relationship")
plt.savefig("age_pregnancy_relationship.png")
plt.show()
```



The following three cells suggest that pregnant women, likely due to a variety of confounding factors, are not represented in this dataset for any severe effects of COVID-19.

[8]:
```
# Filter the dataframe for female, pregnant patients
pregnant_women = df[(df["SEX"] == 2) & (df["PREGNANT"] == 1)]

# Check if any of these patients died (i.e. have a value other than␣
 ↪"9999-99-99" in the "DATE_DIED" column)
any_pregnant_women_died = pregnant_women["DATE_DIED"].apply(lambda x: x !=␣
 ↪"9999-99-99").any()

# Print the result
print(f"Did any pregnant women die? {any_pregnant_women_died}")
```

```
Did any pregnant women die? False
```

[9]:
```
# Filter the dataframe to include only pregnant women
df_pregnant = df[(df["SEX"] == 2) & (df["PREGNANT"] == 1)]

# Determine if any pregnant woman was intubated
```

```
intubated_pregnant = df_pregnant[df_pregnant["INTUBED"] == 1]

# Check if there are any rows in the intubated_pregnant dataframe
if intubated_pregnant.shape[0] > 0:
    print("There are pregnant women who were intubated.")
else:
    print("There are no pregnant women who were intubated.")
```

```
There are no pregnant women who were intubated.
```

[10]:
```
# Filter the dataframe to include only pregnant women
df_pregnant = df[(df["SEX"] == 2) & (df["PREGNANT"] == 1)]

# Check if any pregnant women went to the ICU
if any(df_pregnant["ICU"] == 1):
  print("There were pregnant women who went to the ICU.")
else:
  print("There were no pregnant women who went to the ICU.")
```

```
There were no pregnant women who went to the ICU.
```

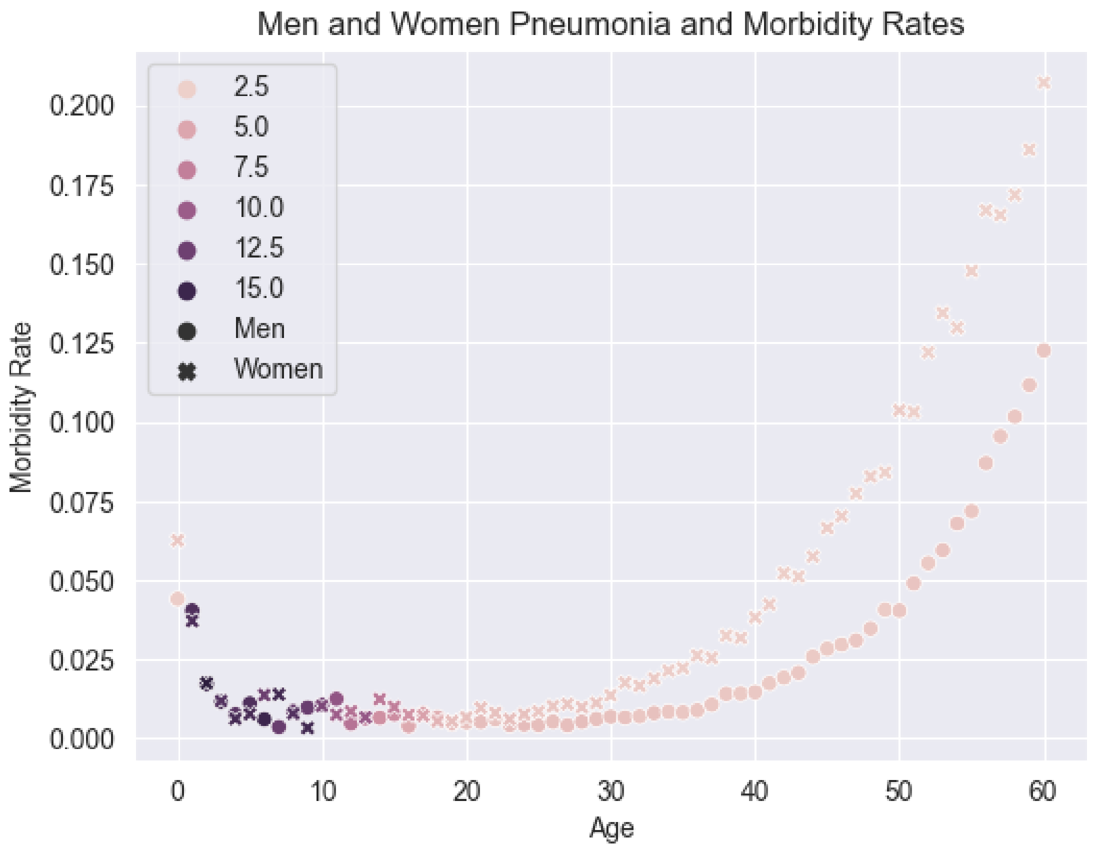
However, there are some interesting differences between the sexes in morbidity rates and their relationship to age.

[11]:
```
# men plot
df_men = df[df["SEX"] == 1]
grouped_men = df_men.groupby("AGE")
death_percentages_men = grouped_men["DEATH"].mean()
pneumonia_percent_men = grouped_men["PNEUMONIA"].mean()

# Non-pregnant women plot
df_women = df[(df["SEX"] == 2) & (df["PREGNANT"] != 1)]
grouped_women = df_women.groupby("AGE")
death_percentages_women = grouped_women["DEATH"].mean()
pneumonia_percent_women = grouped_women["PNEUMONIA"].mean()

# Combine data for men and women
death_percentages = pd.concat([death_percentages_men, death_percentages_women])
pneumonia_percent = pd.concat([pneumonia_percent_men, pneumonia_percent_women])
sex = ["Men"] * len(death_percentages_men) + ["Women"] *␣
 ↪len(death_percentages_women)

# Plot combined data
sns.scatterplot(x=death_percentages.index, y=death_percentages.values,␣
 ↪hue=pneumonia_percent.values, style=sex)
plt.title("Men and Women Pneumonia and Morbidity Rates")
plt.xlabel("Age")
plt.ylabel("Morbidity Rate")
plt.savefig("male_female_pneumonia_fig.png")
plt.show()
```

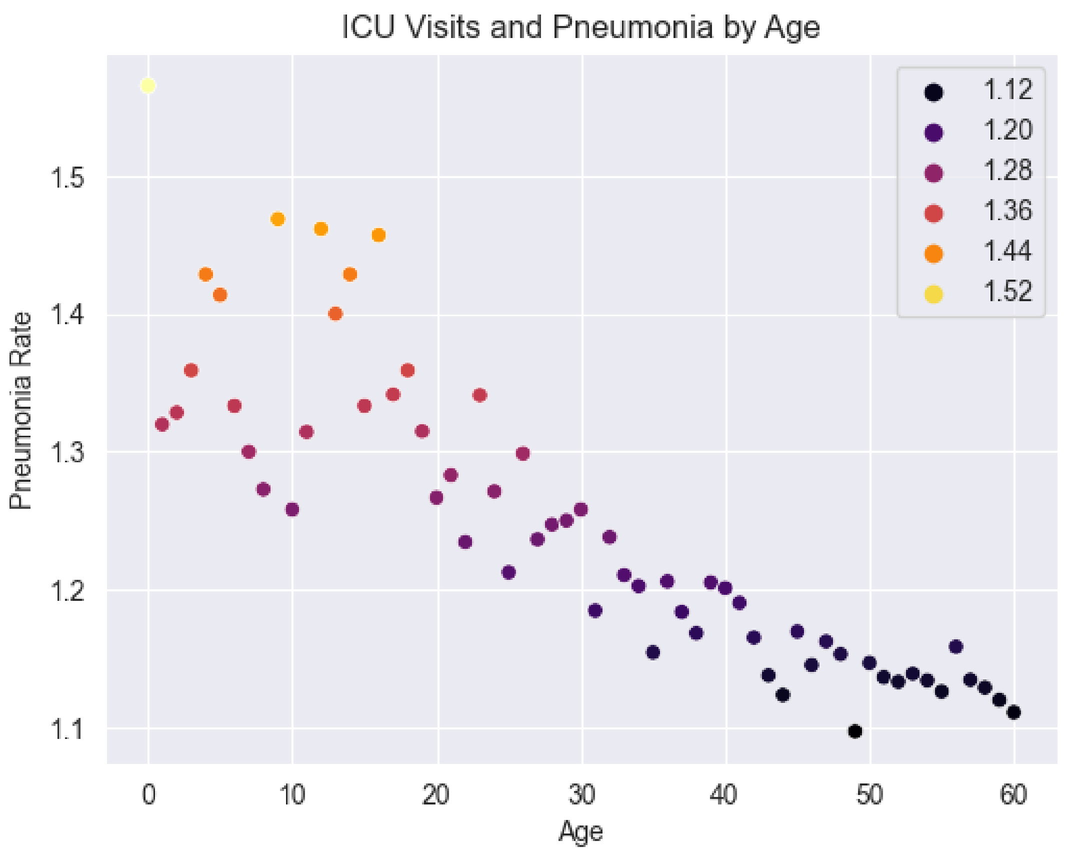


The graph above and the two below would indicate that pneumonia was a much more lethal factor in those youth who visited the ICU or died of COVID-19.

[12]:
```
# Filter the dataframe to include only patients who visited the ICU
df_icu = df[df["ICU"] == 1]

# Group the data by age and calculate the percentage of patients with pneumonia
grouped_icu = df_icu.groupby("AGE")
pneumonia_percent_icu = grouped_icu["PNEUMONIA"].mean()

# Create the scatterplot
sns.scatterplot(x=pneumonia_percent_icu.index, y=pneumonia_percent_icu.values,␣
  ↪hue=pneumonia_percent_icu.values, palette="inferno")
plt.xlabel("Age")
plt.ylabel("Pneumonia Rate")
plt.title("ICU Visits and Pneumonia by Age")
plt.savefig("icu_pneumonia_fig.png")
plt.show()
```

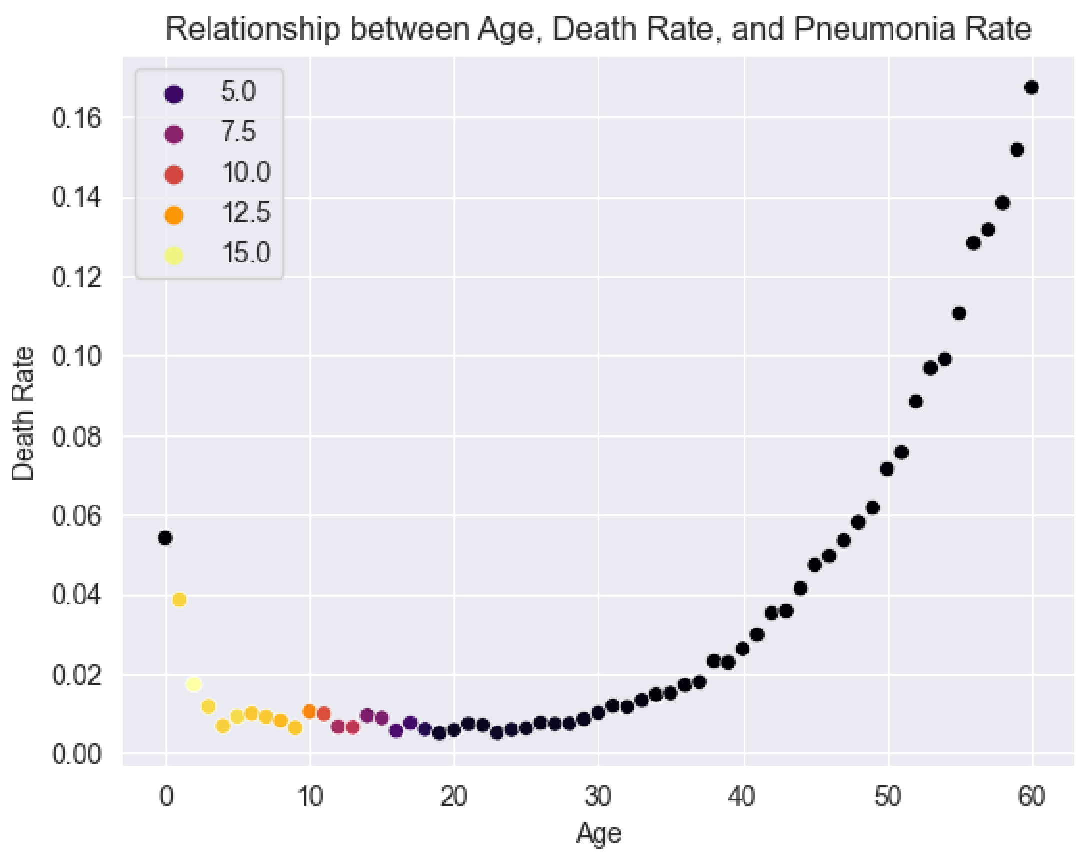


[13]:
```
# Group the data by age
grouped = df.groupby("AGE")

# Calculate the mean death rate and mean pneumonia rate for each age group
death_rates = grouped["DEATH"].mean()
pneumonia_rates = grouped["PNEUMONIA"].mean()

# Create the scatter plot using the mean death rate and pneumonia rate values
sns.scatterplot(x=death_rates.index, y=death_rates.values, hue=pneumonia_rates.
 ↪values, palette="inferno")

# Add labels to the plot
plt.xlabel("Age")
plt.ylabel("Death Rate")
plt.title("Relationship between Age, Death Rate, and Pneumonia Rate")
plt.savefig("age_pneumonia_fig.png")
plt.show()
```



Here is the model analysis. All the models performed well and the random forest is easiest for feature extraction.

[14]:
```
# RF evaluation
y_pred = forest.predict(X_test)
report = classification_report(y_test, y_pred, output_dict=True)
accuracy_list.append(report["accuracy"])
precision_list.append(report["1"]["precision"])
recall_list.append(report["1"]["recall"])
f1_list.append(report["1"]["f1-score"])
print(report)
```

```
{'0': {'precision': 0.9638494318181818, 'recall': 0.9769584779534645,
'f1-score': 0.9703596828355826, 'support': 194475}, '1': {'precision':
0.64422389837237, 'recall': 0.5324146981627297, 'f1-score': 0.5830070055685289,
'support': 15240}, 'accuracy': 0.944653458264788, 'macro avg': {'precision':
0.8040366650952759, 'recall': 0.7546865880580971, 'f1-score':
0.7766833442020558, 'support': 209715}, 'weighted avg': {'precision':
0.9406222276138371, 'recall': 0.944653458264788, 'f1-score': 0.9422107435534621,
'support': 209715}}
```

[15]:
```
# logistic regression model
model = LogisticRegression(solver="lbfgs", random_state=42)
model.fit(X_train, y_train)
```

```
C:\Users\edwin\OneDrive - University of Denver\Personal
Programming\venv\lib\site-packages\sklearn\linear_model\_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[15]:
```
LogisticRegression(random_state=42)
```

[16]:
```
# LR evaluation
y_pred = model.predict(X_test)
report = classification_report(y_test, y_pred, output_dict=True)
accuracy_list.append(report["accuracy"])
precision_list.append(report["1"]["precision"])
recall_list.append(report["1"]["recall"])
f1_list.append(report["1"]["f1-score"])
print(report)
```

```
{'0': {'precision': 0.9536577541179183, 'recall': 0.9782697004756395,
'f1-score': 0.9658069543542765, 'support': 194475}, '1': {'precision':
0.5865375207905293, 'recall': 0.3933727034120735, 'f1-score':
0.4709163033659322, 'support': 15240}, 'accuracy': 0.935765205159383, 'macro
avg': {'precision': 0.7700976374542239, 'recall': 0.6858212019438565,
'f1-score': 0.7183616288601044, 'support': 209715}, 'weighted avg':
{'precision': 0.9269791075933044, 'recall': 0.935765205159383, 'f1-score':
0.9298432249068724, 'support': 209715}}
```

[17]:
```
# decision tree
model = DecisionTreeClassifier(max_depth=250)
model.fit(X_train, y_train)
```

[17]:
```
DecisionTreeClassifier(max_depth=250)
```

[18]:
```
# DT evaluation
y_pred = model.predict(X_test)
report = classification_report(y_test, y_pred, output_dict=True)
accuracy_list.append(report["accuracy"])
precision_list.append(report["1"]["precision"])
recall_list.append(report["1"]["recall"])
```

```
f1_list.append(report["1"]["f1-score"])
print(report)
```

```
{'0': {'precision': 0.9612399561723886, 'recall': 0.9743925954492866,
'f1-score': 0.9677715896642332, 'support': 194475}, '1': {'precision':
0.604102074886716, 'recall': 0.4986220472440945, 'f1-score': 0.5463172651784751,
'support': 15240}, 'accuracy': 0.9398183248694657, 'macro avg': {'precision':
0.7826710155295522, 'recall': 0.7365073213466905, 'f1-score':
0.7570444274213541, 'support': 209715}, 'weighted avg': {'precision':
0.9352867276918618, 'recall': 0.9398183248694657, 'f1-score':
0.9371444818981556, 'support': 209715}}
```

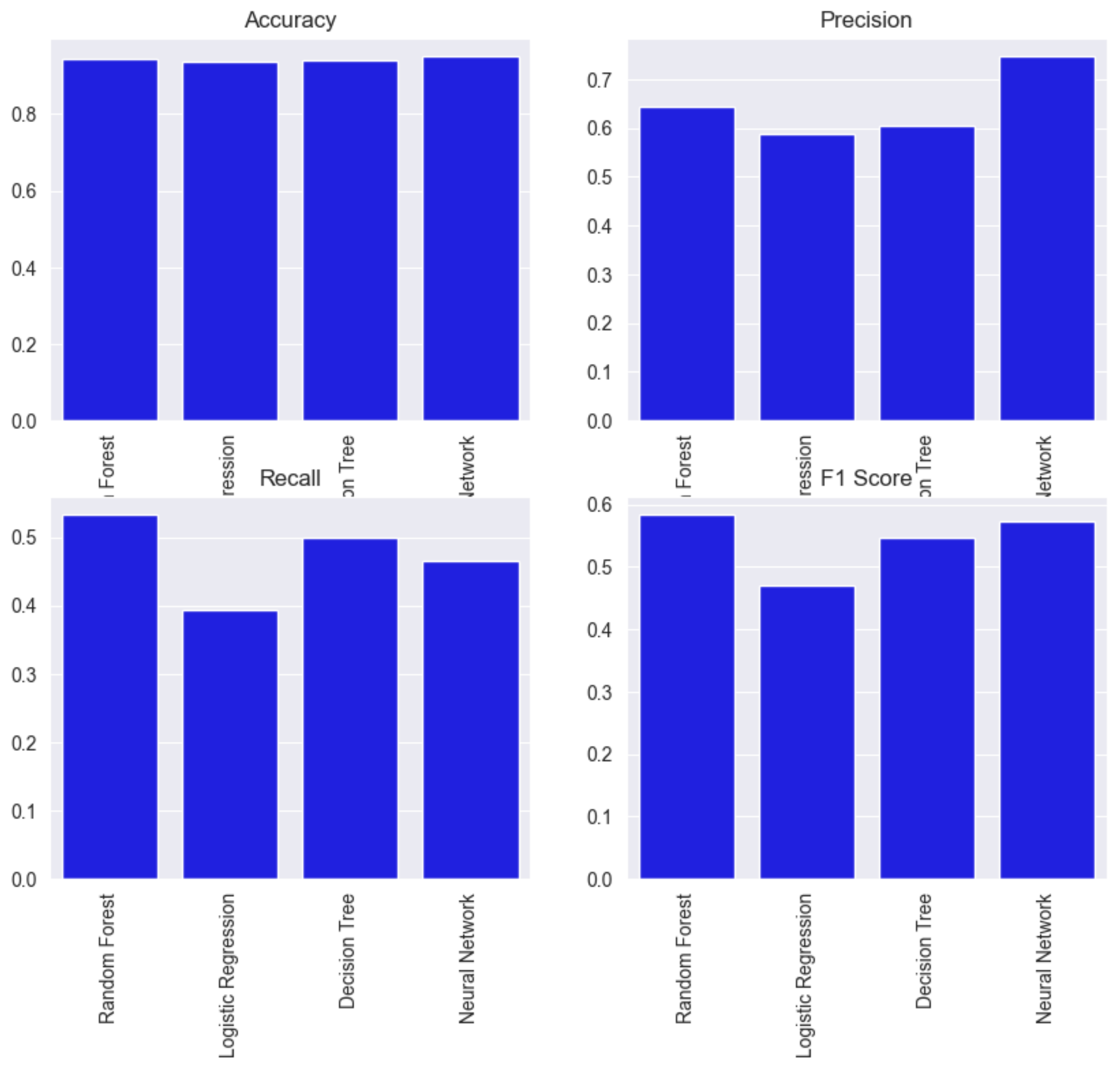
[19]:
```
# NN creation
model = MLPClassifier(hidden_layer_sizes=(10, 10))
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

[20]:
```
# NN evaluation
y_pred = model.predict(X_test)
report = classification_report(y_test, y_pred, output_dict=True)
accuracy_list.append(report["accuracy"])
precision_list.append(report["1"]["precision"])
recall_list.append(report["1"]["recall"])
f1_list.append(report["1"]["f1-score"])
print(report)
```

```
{'0': {'precision': 0.9592879512114719, 'recall': 0.9875819514076359,
'f1-score': 0.97322935194053, 'support': 194475}, '1': {'precision':
0.7458964646464646, 'recall': 0.4651574803149606, 'f1-score':
0.5729873908826383, 'support': 15240}, 'accuracy': 0.9496173378156069, 'macro
avg': {'precision': 0.8525922079289683, 'recall': 0.7263697158612983,
'f1-score': 0.7731083714115841, 'support': 209715}, 'weighted avg':
{'precision': 0.9437807807408299, 'recall': 0.9496173378156069, 'f1-score':
0.9441437477323319, 'support': 209715}}
```

[21]:
```
# comparison plot
model_names = ["Random Forest", "Logistic Regression", "Decision Tree", "Neural␣
 ↪Network"]
num_models = len(model_names)
fig, ax = plt.subplots(2, 2, figsize=(10, 8))
sns.barplot(x=model_names, y=accuracy_list, ax=ax[0, 0], color="b")
ax[0, 0].set_title("Accuracy")
ax[0, 0].set_xticklabels(model_names, rotation=90)
sns.barplot(x=model_names, y=precision_list, ax=ax[0, 1], color="b")
ax[0, 1].set_title("Precision")
ax[0, 1].set_xticklabels(model_names, rotation=90)
sns.barplot(x=model_names, y=recall_list, ax=ax[1, 0], color="b")
ax[1, 0].set_title("Recall")
```

```
ax[1, 0].set_xticklabels(model_names, rotation=90)
sns.barplot(x=model_names, y=f1_list, ax=ax[1, 1], color="b")
ax[1, 1].set_title("F1 Score")
ax[1, 1].set_xticklabels(model_names, rotation=90)
plt.savefig("comparison_plot.png")
```


Accuracy
Precision
Recall
F1 Score
Random Forest
Logistic Regression
Decision Tree
Neural Network